矩阵乘

2019年3月27日 9:40

一. 猫猫吃罐头（USSTD7B）

1. 操作1给第x猫+1个罐头，操作2给第x猫罐头清零，操作3交换xy猫的罐头数
2. 操作1给第x行的第0列+1即可（第0行第0列是常数1）；操作2给第i行全设为0即可；操作3交换x行y行即可
3.
```
const int MN = 1e2 + 5;
int n,m,k;
int op,x,y;
struct Matrix{
    ll v[MN][MN];
    Matrix(){          //默认构造全零
        ms(v,0);}
    inline void set1(){                    //设为单元阵
        for__(i,0,n)
            v[i][i] = 1; }
//      void show(){
//          for__(i,0,n)
//              for__(j,0,n)
//                  printf("%3lld%c",v[i][j]," \n"[j==n]);//     }
    Matrix operator=(Matrix r){       //更新值为另一个矩阵
        for__(i,0,n)
            for__(j,0,n)
                v[i][j] = r.v[i][j];}
    Matrix operator*=(Matrix r){     //乘以另一个矩阵，并更新值
        Matrix t;
        for__(i,0,n)
            for__(j,0,n)
                for__(k,0,n)
                    t.v[i][j] += v[i][k] * r.v[k][j];
        *this = t;}
    Matrix operator^=(int n){          //快速求n次幂
        Matrix t;
        t.set1();//构造单位阵
        Matrix a = *this;
        while(n){
            if(n&1)
                t*=a;
            a*=a;
            n>>=1;}
        *this = t; }
    inline void set0(int I){    //将第I行设0
        for__(j,0,n)
            v[I][j] = 0;}
    inline void add1(int I){   //v[0][0]永远是1，用于实现单点+1
        ++v[I][0];}
    inline void swp(int i,int I){          //交换i行和I行
        for__(j,0,n)
            swap(v[i][j],v[I][j]);}};
```

```
int main(int argc, char** argv) {
    scanf("%d%d%d",&n,&m,&k);
    Matrix ans;
    ans.set1();        //初值为单元阵
    for_(i,0,k){
        scanf("%d%d",&op,&x);
        if(op==1)
            ans.add1(x);
        else if(op==2)
            ans.set0(x);
        else
            scanf("%d",&y),
            ans.swp(x,y);
//              ans.show();
    }
    ans ^= m;
//    ans.show();
    for__(i,1,n)
        printf("%lld%c",ans.v[i][0]," \n"[i==n]);
    return 0;}
```

二. 纯数学题例

对于给定的n与x，请你计算

1. $$f(n)=\sum_{i=1}^{n}(i^x \cdot x^i)$$

2. x=1时套求和公式，x=2时思路如下：
   i. 让第n项减k倍的的n-1项，直至商没有n为止，再逆推
   ii. 令g(n-1)=f(n)-f(n-1)=n^2 * 2^n，则f(n)=f(n-1)+g(n-1)，其中g(n)=2 * (n+1)^2 * 2^n
   iii. 令h(n-1)=g(n)-2g(n-1)=(4n+2)* 2^n，则g(n)=2g(n-1)+h(n-1)，h(n)=2 * (4n+6)* 2^n
   iv. 令i(n-1)=h(n)-2h(n-1)=8* 2^n，则h(n)=2h(n-1)+i(n-1)，其中i(n)=16* 2^n
   v. 令f(0)g(0)h(0)i(0)为初始向量，乘以以下转移矩阵的n次幂即可，或者令f(1)g(1)h(1)i(1)为初始向量，对n-1次方做快速幂

	f(n+1)	g(n+1)	h(n+1)	i(n+1)
f(n)	1			
g(n)=2* (n+1)^2 * 2^n	1	2		
h(n)=2 * (4n+6)* 2^n		1	2	
i(n)=16* 2^n			1	2

3. 另外，令新维度项的时候，也可以直接猜通项，再解通项的k倍，可以得到城爹这种式子：

	f(n+1)	g(n+1)	h(n+1)	i(n+1)
f(n)	1			
g(n)=n^2 * 2^n	1	2		
h(n)=n * 2^n		4	2	
i(n)=2^n		2	2	2

4. 让第n项减k倍的的n-1项，直至商没有n为止，再逆推
5. 注意：传二维数组首址给函数后，sizeof(首址)可能只有指针数组的大小，果然还是需要靠sizeof(类型)*维数来memset

三. 求满足$\arctan(1/x)=\arctan(1/y)+\arctan(1/z)$ ($x < y < z$, $x+y=z$)的第n对正数对(x,y)及x+y

1. 其实这是斐波那契的一个性质
2. 先来一个比较慢，但是有看头的版本
3.
```
ull m0[2][2]={    0, 1,
                  1, 1 };

void mmul(ull m[2][2], ull m2[2][2]){      //m *= m2
    ull mt[2][2];
    memset(mt, 0, sizeof(mt));
    for_(i,0,2)
        for_(j,0,2)
            for_(k,0,2)
                mt[i][j] = (mt[i][j] + m[i][k] * m2[k][j]) % p;
    for_(i,0,2)
        for_(j,0,2)
            m[i][j]= mt[i][j];}

void mpow(ull m[2][2], ull n){     //m = m^n
    ull mt[2][2]={    0, 1,
                      1, 1};
    //此处：mt00=第n项 mt01=mt10=第n+1项 mt11=第n+2项
    while(n){
        if(n&1)
            mmul(mt, m);
        mmul(m, m);
        n>>= 1;}
    for_(i,0,2)
        for_(j,0,2)
            m[i][j]= mt[i][j];}

int main(int argc, char *argv[]) {
    int t;
    scanf("%d",&t);
    ull m[2][2];
    ull n;
    while(t--){
        for_(i,0,2)
            for_(j,0,2)
                m[i][j]= m0[i][j];
        scanf("%llu",&n);
        n*=2;
        mpow(m, n);
```

```
        printf("%d %d %d\n", m[0][0], m[0][1], m[1][1]);}}
```

4. 接下来是较快的给n次方转移矩阵打表的，向量矩阵乘法的方法

```
int p= 1000000007;

int v0[2]={     0, 1 };
int m0[2][2]={  0, 1,
                1, 1 };

int mn[70][2][2];                   //mn[i] = m0^i

void vmulm(int v[2], int m[2][2]){          //v *= m
    int vt[2];
    memset(vt, 0, sizeof(vt));
    for_(j,0,2)
        for_(k,0,2)
            vt[j] = (vt[j] + (ull)v[k] * m[k][j]) % p;
    for_(j,0,2)
        v[j]= vt[j];}

void m2(int m[2][2]){   //m *= m
    int mt[2][2];
    memset(mt, 0, sizeof(mt));
    for_(i,0,2)
        for_(j,0,2)
            for_(k,0,2)
                mt[i][j] = (mt[i][j] + (ull)m[i][k] * m[k][j]) % p;
    for_(i,0,2)
        for_(j,0,2)
            m[i][j]= mt[i][j];}

void vmulm0pow(int v[2], ull n){//v *= m0^n
    //v0=第n项 v1=第n+1项
    int t= 1;
    while(n){
        if(n&1)
            vmulm(v, mn[t]);
        n>>= 1;
        ++t;}}

int main(int argc, char *argv[]) {
    for__(t,1,64){
        for_(i,0,2)
            for_(j,0,2)
                mn[t][i][j]= m0[i][j];
        m2(m0);}
    int t;
    scanf("%d",&t);
    int m[2][2];
    ull n;
    int v[2];
    while(t--){
        for_(j,0,2)
            v[j]= v0[j];
        scanf("%llu",&n);
        n*=2;
        vmulm0pow(v, n);
```

```
printf("%d %d %d\n", v[0], v[1], (v[0] + v[1]) % p);}}
```

四. 求斐波那契第a^b项对n取余的余数

1. 算a^b似乎并不能对n取余，因为第a^b对n取余项一般和a^b项无关
2. 不过斐波那契的后几位，是隔几百几千项会循环的，所以对n取余的余数在n方次内暴力搜应该能搜到一轮循环需要的次数loop
3. 让a^b对loop取余即可
4.
```
ull a,b;
int t,n,loop;

ull qpow(ull a,ull n){
    ull ans=1;
    for(;n;n>>=1){
        if(n&1)
            ans=ans*a%loop;
        a=a*a%loop;}
    return ans;}

void mul(ull f[2],ull m[2][2]){
    ull f2[2];
    memcpy(f2,f,sizeof(ull)*2);
    f[0]=(f2[0]*m[0][0]+f2[1]*m[1][0])%n;
    f[1]=(f2[0]*m[0][1]+f2[1]*m[1][1])%n;}

void sq(ull m[2][2]){
    ull m2[2][2];
    memcpy(m2,m,sizeof(ull)*4);
    memset(m,0,sizeof(ull)*4);
    for_(i,0,2)
        for_(j,0,2)
            for_(k,0,2)
                m[i][j]=(m[i][j]+m2[i][k]*m2[k][j]%n)%n;}

int main(){
    cin>>t;
    while(t--){
        cin>>a>>b>>n;
        if(n==1){           //怎么取余都是0
            cout<<0<<endl;
            continue;}
        ull ft[2]={1,1};
        for__(i,3,n*n){
            ft[i&1]=(ft[i&1]+ft[1-(i&1)])%n;
            if(ft[i&1]==1 && ft[1-(i&1)]==0){
                loop=i-1;
                break;}}
        ull f[2]={1,0};
        ull m[2][2]={1,1,1,0};
        for(ull N=qpow(a%loop,b%loop);N;N>>=1){
            if(N&1)
                mul(f,m);
            sq(m);}
        cout<<f[1]<<endl; }
    return 0;}
```

五. 裸函数模板

```
1. void vmulm(int v[2], int m[2][2]){          //v *= m
        int vt[2];
        memset(vt, 0, sizeof(vt));
        for_(j,0,2)
            for_(k,0,2)
                vt[j] = (vt[j] + (ull)v[k] * m[k][j]) % p;
        for_(j,0,2)
            v[j]= vt[j];
   //     printf("\nvmulm ");
   //     for_(j,0,2)
   //         printf("%llu ",v[j]);
   }
2. void m2(int m[2][2]){    //m *= m
        int mt[2][2];
        memset(mt, 0, sizeof(mt));
        for_(i,0,2)
            for_(j,0,2)
                for_(k,0,2)
                    mt[i][j] = (mt[i][j] + (ull)m[i][k] * m[k][j]) % p;
        for_(i,0,2)
            for_(j,0,2)
                m[i][j]= mt[i][j];
   //     printf("\nm2 ");
   //     for_(i,0,2)
   //         for_(j,0,2)
   //             printf("%llu ",m[i][j]);
   }
3. void mmul(int m[2][2], int m2[2][2]){    //m *= m2
        int mt[2][2];
        memset(mt, 0, sizeof(mt));
        for_(i,0,2)
            for_(j,0,2)
                for_(k,0,2)
                    mt[i][j] = (mt[i][j] + (ull)m[i][k] * m2[k][j]) % p;
        for_(i,0,2)
            for_(j,0,2)
                m[i][j]= mt[i][j];
   //     printf("\nmmul ");
   //     for_(i,0,2)
   //         for_(j,0,2)
   //             printf("%llu ",m[i][j]);
   }
4. void vmulm0pow(int v[2], ull n){//v *= m0^n
        while(n){
            if(n&1)
                vmulm(v, m);
            m2(m);
            n>>= 1;
        }
   }
5. void vmulmpow(int v[2], int m[2][2], ull n){        //v *= m^n
        while(n){
            if(n&1)
                vmulm(v, m);
            m2(m);
            n>>= 1;
        }
```

```
}
6. void mpow(int m[2][2], ull n){    //m = m^n
        int mt[2][2]={    0, 1,
                                        1, 1};
        //斐波那契mt00=第n项 mt01=mt10=第n+1项 mt11=第n+2项
        while(n){
            if(n&1)
                mmul(mt, m);
            m2(m);
            n>>= 1;
        }
        for_(i,0,2)
            for_(j,0,2)
                m[i][j]= mt[i][j];
//      printf("\nmpow ");
//          for_(i,0,2)
//              for_(j,0,2)
//                  printf("%llu ",m[i][j]);
}
```

六. class模板

1. 虽然还是根据题目静态开更好，但写都写了就贴这吧.....
2. 矩阵平方：
    i.
```
struct Mat{
    int r,c;
    ull**m;
    Mat(int r,int c):r(r),c(c){
        m=new ull*[r];
        for_(i,0,r)
            m[i]=new ull[c]();}
    void square(){
        ull **t=new ull*[r];
        for_(i,0,r)
            t[i]=new ull[c]();
        for_(i,0,r)
            memcpy(t[i],m[i],sizeof(ull)*c);
        for_(i,0,r)
            memset(m[i],0,sizeof(ull)*c);for_(i,0,r)
            for_(j,0,c)
                for_(k,0,c)
                    m[i][j]=(m[i][j]+(ull)
                        (t[i][k]*t[k][j]))%P;
        for_(i,0,r)
            delete[]t[i];
        delete[]t;}};
```
3. 向量乘矩阵：
    i.
```
struct Vec{
    int d;
    ull *v;
    Vec(int d):d(d){
        v=new ull[d]();}
    void mul(Mat m){
        ull *a=new ull[d];
        for_(i,0,d)
            a[i]=v[i];
        memset(v,0,sizeof(ull)*d);
```

```
for_(j,0,d)
    for_(k,0,d)
        v[j]=(v[j]+(ull)a[k]*m.m[k][j])%P;
delete[]a;}};
```

4. 向量乘矩阵快速幂：

i.
```
for(;n;n>>=1){
    if(n&1)
        v.mul(m);
    m.square();}
```